Turntaking:
a case for agent-based programming

Joris Hulstijn
Gerard A.W. Vreeswijk

institute of information and computing sciences, utrecht university

technical report UU-CS-2003-045

www.cs.uu.nl

Turntaking: a case for agent-based programming

Joris Hulstijn
Gerard A.W. Vreeswijk
ICS, Utrecht University

December 17, 2003

Abstract

In this report we compare two programming languages for the development of agent-based programs. We compare 3APL, an abstract agent programming language, with Ruby, an object-oriented scripting language. The test case we selected is the turn-taking problem: how to allocate turns between speaking agents that communicate by means of an exclusive speech channel. We show that both languages are sufficiently equipped to implement a solution to the turn taking problem. Ruby is faster, and more fun to use. Ruby supports parallel processing. 3APL supports higher level concepts, and therefore does not directly fit this test case. Although the 3APL agent platform allows a natural model of a dynamic participants list, we find that the development of 3APL is far from complete. Based on our experiments, we conclude that the syntax needs to be re-evaluated and the internal deliberation cycle should be changed so as to achieve a true event-based deliberation cycle.

Contents

1 Introduction

Agent-based programming is rapidly gaining status as a new paradigm for programming autonomous systems. After the object-oriented programming paradigm in which data and methods are clustered into objects, agent-based programming has relegated the control to the agents themselves. Where objects can only react, agents can in principle pursue their own goals. Due to this autonomy of agents, both with respect to knowledge and control, agent-based systems are potentially more efficient and more robust to change than centralized systems. But these benefits come at a cost: the distributed nature of the problem makes the solution inherently more complex. To overcome this additional complexity, a programmer will need specialized tools and programming languages for agent-based programs [17]. Thus, besides agent-based modeling techniques, also implementation aspects have been addressed.

Starting from the language Agent0 developed by Shoham [29] various agent-based methodologies and programming languages have been developed.[1] Agent-based programming languages or methodologies are essentially based on a programming metaphor that makes use of the mental attitudes of agents, like beliefs, goals and plans. It is claimed that such high-level concepts make it easier to model and implement solutions to intelligent tasks. Here we mention the agent-based methodology DESIRE [5] and the cognitive agent architecture SOAR [19], used to generate psychological predictions. Moreover, tasks which are inherently distributed, or that can be sub-divided in smaller tasks, can be solved more efficiently by groups of agents in cooperation. Methodologies for the design of such multi-agent systems often make use of concepts from the social sciences, in particular of organization models. Examples of multi-agent methodologies are Gaia [43], TROPOS [6] and OperA [11]. Environments used for simulation experiments with multi-agent systems are SWARM [14] and SGML [21]. Examples of dedicated agent-based programming languages are Concurrent MetateM, an executable temporal logic [13]; PLACA, an extension of Agent0 [36], AgentSpeak [25] and our 3APL [15]. Recent developments have focused on platforms for agent-based programming, such as JADE [3], RETSINA [31]. These platforms provide facilities for agent management, agent communication, and directory services.

One of the main problems in agent programming, is the large gap between theoretical work, often inspired on the BDI models of agency [8, 27, 26] and practical work. Our own research on 3APL, An Abstract Agent Programming Language, [15, 10, 9, 38] has tried to bridge this gap. On the one hand 3APL is a real programming language; on the other hand it has a formal semantics, which connects the programming constructs – beliefs, goals and practical reasoning rules – to BDI theories of agency. Further research effort seeks to develop the strengths of 3APL as a practical programming language. Recently, 3APL has been extended with an agent development platform, which provides a directory service and allows agent communication by means of message passing [10]. 3APL has also been adapted for robotics applications, with sensing actions [39]. However, to date, the language 3APL has not been tested on a real-time multi-agent application.

In this paper we report on the application of the 3APL agent-based programming language to the *turn-taking* problem [28]. Turntaking mechanisms are found in all area's where human or software agents communicate synchronously through a limited number of communication channels. The prototypical example of turntaking is a discussion within a small group such that participants compete for speaking time. Other examples are a relay race, waiting in line at a counter, or limiting speaking time in parliament. In computer science, turn-taking and exclusion algorithms have been studied in the areas

[1] A regularly updated overview is maintained by Agent Link, `http://www.agentlink.org`.

of concurrent programming and distributed problem solving [34, 33, 32]. In this paper we focus on human turn-taking mechanisms. We take as a starting point the account of Harvey Sacks et al [28]. This research originated in conversation analysis, which is a sub-field of sociolinguistics. The account provides a number of basic observations, collected during careful analysis of dialogue transcripts. The observations can be explained by a relatively simple turn taking mechanism. The mechanism is described in a abstract way, which – after some additional interpretation – allows it to be implemented in a computer simulation. Other simulations have been done of turn taking between two participants [24, 16] and in groups [22]. However, these approaches focus on the empirical accuracy of predicting the distribution of turns in human dialogue; we focus on the computational aspects.

By itself, a computational account of turn-taking already has some benefit. Computational turn-taking mechanisms are useful for the design and evaluation of mixed-initiative dialogue systems [4]. In such systems, both system and user may take the initiative, and solicit contributions to the dialogue. To build such systems, we must understand how initiative can be taken and granted [30]. If moreover we add the possibility of the user interrupting the system, or the system interrupting the user, we need to understand at what places during an utterance interruptions are to be expected or preferred. Consider for example an automated language teacher, that wants to correct a pupil as soon as possible after the mistake has been made [1]. In such cases, a rude early interruption may damage the trustworthiness of the system, but a late interruption, may damage the learning effect of the correction. Therefore the system must be able to monitor the dialogue and project a good moment to take the turn. Other applications of turn taking may lie in the field of computer supported cooperative work. Consider for example a computer mediated conference system, that allows groups of people to have a meeting, without them being present in the same room. The system must be able to understand aspects of turn taking behavior, in order to facilitate the meeting process. Finally, a study of turn taking mechanisms might inspire work in distributed computing [33]. Current algorithms typically make use of a predetermined turn taking order; by contrast human turn taking is distributed.

Our objective is to write a simulation program which produces results that correspond as much as possible to the observations reported by Sacks et al. An agent-based solution in 3APL is compared with a solution written in Ruby [20]. Ruby is an object-oriented programming language that has aspects of script languages like Perl and specific features to deal with concurrency.

We selected turn taking as a test application for 3APL, because it is a real-time distributed communication problem. The problem is non-trivial, but manageable: it can be formalized and simulated without much additional resources. We selected a real-time problem, because this is a hard aspect of programming, where we expect that the benefits of a dedicated programming language ought to show. We selected a distributed problem, because 3APL is advocated as a language for programming multi-agent systems; not just for single agents. We selected a communication problem because of its generality; solutions may be extrapolated to other distributed problems. Note that it is generally assumed in distributed problem solving that the complexity of the processing of individual agents, may be neglected in comparison to the complexity of the message passing [33].

The paper is structured as follows. Section 2 describes in detail the turn taking mechanisms as formulated by Sacks et al. We list possible applications of automated turn-taking. Section 3 discusses the design of a simulation of turn taking behavior. What are the basic data-structures, what assumptions have to be made, and how can these assumptions be motivated? Sections 4 and 5 deal with the implementation aspects of the simulation in Ruby and 3APL respectively. Section 6 lists the results of both simulations. Finally, in the conclusions we try to make a comparison between the agent-based approach in 3APL and the approach using Ruby.

Manifestations of turntaking

1. Speaker change recurs, or at least occurs.
2. Overwhelmingly, one party talks at a time.
3. Occurrences of more than one speaker at a time are common, but brief.
4. Transitions with no gap or overlap are common; together with transitions with a slight gap or overlap they form the majority of transitions.
5. Turn order is not fixed, but varies.
6. Turn size is not fixed, but varies.
7. Length of conversation is not fixed in advance.
8. What parties say is not fixed in advance.
9. Relative distribution of turns is not specified in advance.
10. Number of parties can vary.
11. Talk can be continuous, or discontinuous.
12. Turn allocation techniques are obviously used. Either the speaker selects the next speaker by addressing him or her, or speakers may self-select.
13. Various turn-constructional units are employed (word, phrase, sentence ..)
14. Repair mechanisms exist for dealing with turn-taking errors and violations. In particular, if two parties find themselves talking at the same time, one of them will stop.

Table 1: Manifestations of turntaking as observed by Sacks *et al.* [28].

2 Sacks *et al.'s* turntaking theory

Sacks *et al.* developed a theory of turntaking that to date is authoritative in the area of discourse analysis [2, 18, 23, 41]. A summary of their work is published in their review article "A simplest systematics for the organization of turn-taking for conversation" [28]. This research is an example of conversation analysis, which is rooted in ethnomethodology. In this part of sociology, researchers try to make observations of human social behavior, without prejudice or theoretic interpretation. Only after collecting the data, generalizations are allowed. As a consequence, linguistic categories (noun, verb) are not treated in a special way. Researchers tend to focus on the process of speaking, and not on what is being said. Unlike traditional linguists, researchers from conversation analysis therefore tend to stress non-verbal aspects of communication, such as the influence of eye-gaze and non-verbal feedback. Recently, findings from conversation analysis have been integrated with linguistically informed theories of dialogue, and results from psycholinguistics. See Clark [7] for an overview.

The elementary building brick of Sacks *et al.*'s theory of turntaking is the concept of a so-called *turn-constructional unit* (TCU). Informally, a TCU is a stretch of speech, at the end of which another person could (but need not) start speaking. Typically TCU's are complete units with respect to the intonation contour, the grammar, and the semantics. The term TCU is almost synonymous with the term utterance. Grammatically, a TCU may be a complete sentence, a phrase or just a word.

The completion of a TCU results in a *transition-relevance place* (TRP). A TRP marks the possibility for another speaker to take over and start speaking. Hearers are often able to project the end of TCU's, using various cues. Such cues may be linguistic, based on the grammatical structure or meaning, or based on the intonation, but cues may also be non-verbal, including gaze and body movement. In this way, interruptions can be planned in such a way that utterances link up seamlessly. One or more TCU's produced by the same speaker constitute a *turn*, and if all goes well a flowing conversation is the result.

Based on empirical research, Sacks *et al.* made a number of observations. These observations are listed in Table 1 on the preceding page.

The differences of these observations with institutionalized turn-taking systems are obvious. For example, in meetings or debates, the turn order is often fixed or determined by the chairman, and the turn size is limited. By denying each of these observations, you get a different kind of interaction. For example, if (1) is false, we have a monologue. If (2) is false, we get chaos, given the fact we still use the speech channel (auditive perception capabilities), which is non-persistent and which does not allow interference. On a non-exclusive, persistent channel as in asynchronous interaction (e.g. blackboard, news, email) we may get cross-postings. But then we need a different way of organizing contributions (e.g. using topics or threads). If (3) is not true, either there are no overlaps, as when speakers have to use a device which guarantees exclusion (e.g. single microphone), or there are overlaps which are not brief, as in asynchronous interaction (news). If (4) is not true, we have large gaps (e.g. asynchronous interaction like old fashioned mail), or we have large overlaps, see above. If (5) is not true, the turn order is fixed, as in a round robin (Dutch: kringgesprek). If (6) is not true, the turn size is fixed, as in political debates. If (7) is false, the length of the conversation is limited, as in television programmes. If (8) is false, what parties say is in fact fixed in advance, as in a play. If (9) is false, relative distribution of turns (or alternatively relative speaking time) are fixed in advance, as in political debates. Clause (10) is ambiguous. It might mean that the number of parties varies among conversations and can have e.g. 4, or 7 participants, or it might mean that the number is changed during the conversation itself by new parties entering the conversation. A denial of the first interpretation are rules where the rules depend on the number of participants, e.g. the 3rd person on the waiting list gets the turn; a denial of the second is that the number of participants is fixed. This is the case in many closed environments, where only admitted members may speak. Clause (11) is strange. Discontinuous talk, means there are pauses (conversation on a bus). Continuous talk means there are no pauses, as e.g. on the radio. It means that the rules do not favor one over the other. If clause (12) is not true Turn allocation techniques are disallowed. This happens for example when some other person than the speaker (teacher) is allocating turns. If (13) is not true, it means we are limited to one kind of turn constructional unit. Take for example a game like 20 questions (Dutch: Ik zie, ik zie wat jij niet ziet) in which you are only allowed to answer with 'yes' or 'no. If (14) is not true, no repair exists. This happens in heated discussions or when participants are not sufficiently aware of each other.

2.1 Overlap and pauses

Two of the most important phenomena in turntaking are *overlap* and *pauses*. An overlap occurs when, at a certain moment, two or more people speak at once, and a pause occurs when, at a certain moment, no one speaks.

Research pointed out that an overlap has different causes. It may happen for instance, that, due to

Rules for turntaking

1. For any turn, at the first transition-relevance place of the first turn constructional unit:
   (a) the speaker may select the next speaker. In this case, the person selected is the only one with the right and obligation to speak,
   (b) else, the next speaker may self-select. The first person to speak acquires the right to a turn,
   (c) else, the current speaker may continue, but need not continue unless someone else self-selects.
2. Rules 1a-c apply recursively for each next transition relevant place, until transition is affected.

Table 2: Rules for turntaking as proposed by Sacks *et al.* [28].

a timing problem, someone misprojects the end of the turn and starts speaking too soon. Another cause of overlap is when a person speaks without ”taking the floor”, usually using a ”continuer” (e.g. ”uh-huh”). This is to let the speaking person know that you are listening. Overlap may also occur when one person helps another by co-producing the end of the turn (a kind of collaboration), or when one person starts speaking before the other is done in order to silence them (a kind of competition).

Pauses have different causes as well. One speaks of a *gap* when floor-change has not yet occurred (it is nobody's turn to speak) while turn-taking roles are applied. A *lapse* occurs when speaker-selection rules are not being applied. Finally, an *attributable silence* occurs when someone has the floor but doesn't speak.

Timing problems have either cognitive or social explanations. Cognitive timing problems occur when potential speakers are busy with activities such as utterance planning and word search. Interactional trouble, such as a dispreferred response, can be categorized as a social timing problem.

2.2 Rules for turntaking

Based on the empirical research described above, Sacks *et al.* distilled a basic mechanism for turntaking of which the rules are copied in Table 2. A transition-relevance place is a moment, indicated by syntax and prosody, where a transition may take place. A turn constructional unit is basically an utterance.

Agent A may non-verbally address other agents in different ways. If A wants to address B without the use of language it may for example look at B, physically turn to B, or move to B. For the sake of simplicity, we assume that all forms of non-verbal addressing are mapped to visual addressing. Thus, rather than saying: “Agent A non-verbally addresses agent B,” we simply say: “A looks at B”.

Turn Period of talking by one person, delimited by turns of other people, or by prolonged silence.

Turn-constructional unit Period of talking, from which turns are constructed. This can be any linguistic unit that is clearly bound: words, phrases, sentences, segments, ..., but also non-verbal communicative acts (nodding, shaking hands, symbolic gestures, ...)

Transition-relevance place Moment between turn constructional units, at which a transition may occur. Transition-relevance places and turn constructional units are determined by some external (linguistic) talk generation process, which produce probable boundaries.

Current speaker According to the formulation, the rules apply "for any turn , at the first transition-relevance place of the first turn constructional unit". So the current speaker is the speaker of the first part of the current turn until the transition-relevance place.

Next speaker The person with the right (speaker self-selects) and obligation(self-select, speaker selects next) to speak at the interval after the transition-relevance place.

Right to speak Access to the speech channel, which is a resource which can only be used by one person at a time.

Obligation to speak The speech channel is a scarce resource. If the next speaker does not use it, it is wasted.

Continue turn Current speaker may continue the turn, with a new turn constructional unit (e.g. utterance, mover in game).

Recursive application This may be confusing. We doubt that Sacks *et al.* are referring to the technical meaning. We suppose they mean: iterative application. In other words, it applies for each new transition-relevance place.

Hearer An agent receiving utterances from the current speaker, not necessarily addressed to that agent.

Addressee Agent, explicitly addressed while receiving utterances from the current speaker; likely to be the next speaker.

Table 3: Concepts in Sacks *et al.*'s theory.

2.3 Multiple agents

Sacks *et al.* [28] have not much to say on multiple participants in particular. The algorithm is in principle capable of dealing with it. In case there are only two participants, the turn-taking problem still exists, since the speaker can prolong his or her turn.

A successful turn taking mechanism becomes relatively more important as the number of participants, and therefore the possibility of interference, increases. Therefore it is not a coincidence that we find explicit turn taking mechanisms in discussion contexts such as parliament or court. Roughly, there are two common ways of regulating turn taking, which can be found both in human and artificial applications. There is a circular one, as for example in a round robin or a token ring approach [32]. In this case, participants are ordered in a ring topology. Each participant has two neighbors. The right to speak is passed around, clockwise or counterclockwise. In this case there is a star shaped topology with one dedicated participant, the chairman, to regulate turn taking. Interestingly, empirical research into the distribution of turn duration and agreement among participants, revealed that there are roughly two models: a dialogue model, in which separate pairwise discussions can be distinguished, and a monologue model, in which a dominant participant emerges, who speaks relatively more than each

of the others. In smaller groups (5 members) the pairwise dialogues prevail, whereas in larger groups (10 members), the monologue model prevails [12].

3 Design

In order to arrive at an executable protocol, we need more details than Sacks *et al.* have to offer. One problem, for example, is to simulate the underlying linguistic generation process that produces turn-constructional units with clear boundaries. Another problem is to come up with a simple mechanism for self-selection and continuation, such that these mechanisms are orthogonal to (i.e., stay out of the way of) Sacks *et al.*'s framework.

Below we summarize the conceptual choices that we made in order to arrive at an executable protocol.

3.1 Medium

The first choice we have to make is the through which utterances are transported. Possible choices here are metaphors that lean on communication media such as table, floor, air, channel, line, blackboard, newsgroup, e-mail, or forum.

1. *Nature of the medium*. Which metaphor do we follow?

   Here is a consideration: in Sacks *et al.* all communication goes by air and does not involve any writing or archiving. Thus, it seems natural to follow some sort of air-metaphor. The name "air" is perhaps somewhat forced or affected, but for now it is the best name we could find.

   Communication by air implies that multiple parties can speak at once. It also implies that the medium is volatile, i.e., that utterances disappear shortly after they are emitted.

2. *Constraints*. Are there constraints on accessibility? In particular: is the medium open to all parties? Are there speaking privileges?

   It partially follows from our previous choice that all participants may speak freely, and hear everything what others say.

3. *Quality of the medium*. Should all messages come through unharmed?

   Our model assumes that communication is without noise.

The resulting model is simple but adequate. The model easily permits extension. We could, for example, introduce different conditions under which participants are allowed to speak. Similar constraints may be put on the reception of messages. Further, it is not difficult to incorporate noise into our model. However, such extensions, interesting though they may be, overshoot the objective of this paper: to simulate a turntaking protocol based on the work of Sacks *et al.*

We arrive at the data-structure displayed in Table 4 on the following page. It works as follows: what just has been said can be get ("heard") from the set "what-just-has-been-said". Anything that is said now is put in the set "what-is-said-now". The idea is that this type of medium is updated at very short time intervals such that "what-just-has-been-said" evaporates, while "what-is-said-now" becomes "what-just-has-been-said". This models the evanescent character of speech as a communication medium [7].

Medium			
field	**type**	**initialization**	**default**
what-just-has-been-said	List	-	empty
what-is-said-now	List	-	empty

Table 4: Data structure for a medium.

Turn-constructional unit			
field	**type**	**initialization**	**default**
sender	Agent	mandatory	-
addressee	Agent	optional	nil
contents	String	optional	empty string
index	Integer	optional	0

Table 5: Data structure for a turn-constructional unit.

3.2 Turn-constructional unit

In the discourse and dialogue literature there has been a debate on the nature of the turn-constructional unit. Nowadays, it has become more common to equate turn-constructional units with utterances, which may then consist of various linguistic units: exclamations, words, phrases or full sentences. These may be separated by pauses, intonational clues or non-verbal signs. Much work on turn-taking has focussed on the question exactly what aspects of an utterance signal a TRP. See [37] for an overview. However, for the purpose of our simulation, the linguistic realization of a TCU and a TRP do not matter. We only need their structure with respect to the algorithm.

A turn-constructional unit is represented by a data-structure as indicated in Table 5. The field "index" indicates as to how far a TCU has been emitted. Initially, it points to the first character of the contents of a TCU.[2] If a speaker decides to address another participant, it can do so in two ways. The first way is to mention the name of the addressee in the TCU's contents, which might be referred to as *verbally addressing* a participant. The other way, then, is to non-verbally address a participant, for example by pointing, looking or nodding. For this reason, we have a specific field in the data-structure to store the addressee, if any. While speaking, an agent should be able to emit a TCU, or rather a TCU's contents, one particle at a time.

3.3 Particle

A particle is a fragment of a TCU that is uttered by an agent, accompanied by information describing

1. the sender
2. the particle's contents
3. whether someone is non-verbally addressed, and
4. the identity of a possible addressee.

[2] In most programming languages, the index of the first element of a string or array is zero. This convention emanates from the C programming language culture.

Particle			
field	**type**	**initialization**	**default**
sender	Agent	mandatory	-
contents	String	mandatory	-
non-verbally addressing	Agent	optional	-

Table 6: Data structure for a particle.

It is no problem when (3) and (4) are represented simultaneously in an implementation. We arrive at Table 6.

Note that a particle does not indicate whether a transmitted TCU is at a transition-relevance place. This is to be determined by the receiver, based on its own language model. A rich language allows many projections, hence many TRPs, while a poor language allows only a few TRPs or no TRPs at all.

3.4 Agent

Upon creation, an agent must receive a name, a language with which it can formulate turn-constructional units, and (a reference to) a medium along which agents can communicate with other agents.

Usually in spoken face-to-face dialogue, participants are aware of each other. They actively maintain a list of who is present. In this prototype we have chosen to make the list of participants known to all participating agents at the beginning, and to keep it fixed. In a more elaborate version, we might add procedures for entering and exiting a group of participants. Further, at any stage in a conversation, an agent should have knowledge of who talks, who is addressed, what it heard from other agents, and what it thinks other agents try to say. The latter has to do with projection. To this end each agent is equipped with four dictionaries (or hashes, or look-up tables) that map agents to various agent properties.

At this point, we will have to make a choice. The problem is that Sacks *et al.*'s original rule set does not say much about the mechanism according to which participants maintain a list of agents indicating to whom they are addressed by. In fact, only the speaker has the right to allocate a turn. Here, we opt for a more detailed construction. We stipulate that each agent is equipped with dictionaries named "mode," "received-from," "projected," and "addressed-by" (Table 7 on the next page). For example, the dictionary "mode" maps other agents to the current speaking mode they are in, which is either "silent" or "speaking". The dictionary "received-from" maps agents to incomplete TCU's that the agent itself receives from other agents. Similarly with the dictionary "projected". The dictionary "addressed-by" maps agents to sets of other agents. For example, if A maintains the associative array

$$\text{addressed-by} = \{\, A \to \{C\},\ B \to \emptyset,\ C \to \{A, D\},\ D \to \emptyset \,\} \tag{1}$$

then A holds it that it is currently addressed by C, that B and D are currently not addressed, and that C is addressed by both A and D.

An overview of an agent's data-structure is presented in Table 7 on the following page.

Agent				
attribute	**type**	**description**	**initialization**	**default**
name	String	My name.	mandatory	-
medium	Medium	By which way do we communicate?	mandatory	-
language	Language	Words I speak and recognize.	mandatory	-
speaking	Boolean	Do I currently speak?	-	false
mode	agents → speaking modes	Who is speaking?	-	empty
received-from	agents → incomplete TCU's	What did I receive thus far?	-	empty
projected	agents → complete TCU's	What did I project thus far?	-	empty
addressed-by	agents → sets of agents	Who is addressed by whom?	-	empty

Table 7: Data-structure of an agent.

3.4.1 Main loop

Traditionally, agents that are based on a BDI-architecture cycle trough a percept-evaluate-act loop [42]. In our case this naturally boils down to a listen-evaluate-speak loop. Most of the work happens in the evaluation part, so we will expand on that first.

The evaluation part of the listen-evaluate-speak loop is divided into two units: one that is concerned with processing what has been heard, and one that is concerned with formulating TCUs and uttering them. (Fig. 1 on the next page). The latter unit is concerned with what one normally does when one speaks (such as emitting TCU's and deciding whether to continue speaking), and one that is concerned with what one normally does when one is silent (deciding whether to start speaking and conceiving TCU's). In any case, all participants listen, whether they speak or not.

A disadvantage of following the traditional listen-evaluate-speak cycle is that each listen-act must be followed by precisely one evaluation act, which in turn must be (or actually: is) followed by precisely one speech-act. In more advanced architectures, these processes would likely be implemented in parallel.

3.4.2 Speaking

The rules of the turntaking protocol show a potential for deadlock. At the point where nobody is speaking, anyone may self select. To avoid interference, one person must give up the attempt. However, if all agents decide to give up, the channel will not be used efficiently and a new deadlock situation may develop. Roughly, there are two ways of breaking the deadlock. One is by using individual characteristics of the agents, such as their *urgency to self-select* and their *urgency to continue*. This solution is explained below. A second solution makes use of time stamps. The person who started speaking first, if only a millisecond, is announced the winner and receives the right to speak. This solution is used in the 3APL implementation and discussed in Section 5.

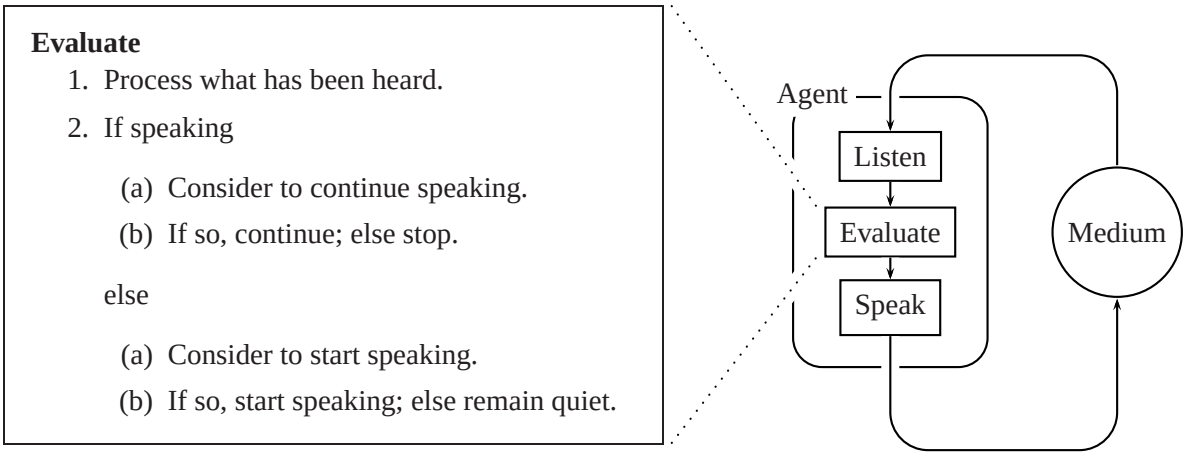

Figure 1: Basic listen-evaluate-speak loop.

Choices at the self-selection phase are the following.

4. *Whether or not to self-select*. If A has the opportunity to self-select, then should it do so? In our model, self-selection depends on the so-called *urgency to self-select*. The urgency to self-select, abbreviated by U, is a numerical value between zero and one. If U is zero, an agent has no urgency to speak at all, and actually will not speak. This actually has to be proven mathematically, but in this case it follows rather straightforwardly from the present model.

   U is determined by two things: the so-called *talk factor* of an agent and the perceived distance to the nearest TRP.

$$U = \frac{\text{TalkFactor}}{1.0 + \text{distance-to-nearest-TRP}^2} \tag{2}$$

   If U is determined, the agent self-selects with a probability of U.

Once an agent has decided to speak, it is confronted with issues that are concerned with TCU-conceptualization.

5. *Whether or not to address another agent*. In the process of forming a TCU, a participant has to decide whether to address someone in particular or to speak to everyone.

   In our current model, this decision is made every time a new TCU is formed and depends on the (for now global) parameter *AddressOtherFactor* $\in [0, 1]$. Agents address other agents with a probability of *AddressOtherFactor*. In such cases the addressee is randomly selected from the agents that are present.

6. *What to say*. In the process of forming a TCU, an agent must to decide what to say. It must decide on the contents of a TCU, and it is specifically addressing another agent, whether to incorporate the name of that other agent into the TCU's contents.

   In our current model, the contents of a TCU depends on whether or not another agent is addressed. If the speaker decided to address another agent, the TCU's contents equals the name of

the addressee with a probability of *MentionOtherFactor* $\in [0, 1]$. In all other cases, the contents of a TCU is randomly selected from a fixed and finite set of strings $\mathcal{L}$. Of course, such a list of strings hardly counts for a language, but fulfills our needs for the purposes of turntaking.

7. *Whether a transition-relevance place is to be put on the TCU, and where to put it*. In the process of forming a TCU, an agent has to determine a point where other agents may interrupt him.

   According to Sacks *et al.*, TRP's by definition mark the end of TCU's. The ambiguity of turntaking is caused by the fact that incomplete TCU's produce multiple projections, rather than that TCU's would have multiple TRP's. Hence, TRP's are placed at the very end of every TCU.

8. *Whether or not to look at another agent while speaking*. If an agent utters a TCU with a non-empty `To:` field, then that agent has to decide, while uttering the TCU, when to non-verbally address (look at) the other agent.

   In our current model, the decision to look at someone is made by measuring the distance to a TRP. We decided that, if the TCU approaches a TRP, the speaker starts non-verbally addressing (i.e., looking at) the addressee. We let implementation decide what it means for a TCU to be close enough to a TRP. See page 17).

9. *Whether or not to continue speaking*. In our model, the decision to continue depends on the so-called *urgency to continue*. The urgency to continue, abbreviated by V (which is a logical successor of U), is a numerical value between zero and one. If V is zero, then the agent has no urgency to continue at all, and will stop speaking immediately. V is determined by the number of agents speaking, in the following way:

$$V = \frac{1.0}{\sqrt{X}} \tag{3}$$

   Thus, V is a discrete, monotonically decreasing, and concave function of X, which basically means that the urgency to continue rapidly decreases as the number of speakers increases. Experimentally, it turned out that $V = 1.0/X$ did not work, because this function would make the urgency to continue too low if more than two speakers were deciding too continue.

   If V is determined, the agent continues speaking with a probability of V.

3.4.3 Listening

In our model, listening involves the processing of particles, matching particles against incomplete TCU's, and projecting incomplete TCU's on well-formed and complete TCU's. Since projection involves some form of guessing, the projection process itself is also non-trivial and open to different forms of implementation. The projections (i.e, completed TCU's) must thereafter be analyzed to their semantics contents, which means that items such as sender, intonation, gesture and contents are located within the TCU and extracted.

10. *When to listen*. Should agents listen unconditionally, or can they choose not to listen?

    Our model assumes that agents listen unconditionally and that communication is without noise. Both complications can very easily be built into our model, but we have to draw a line somewhere.

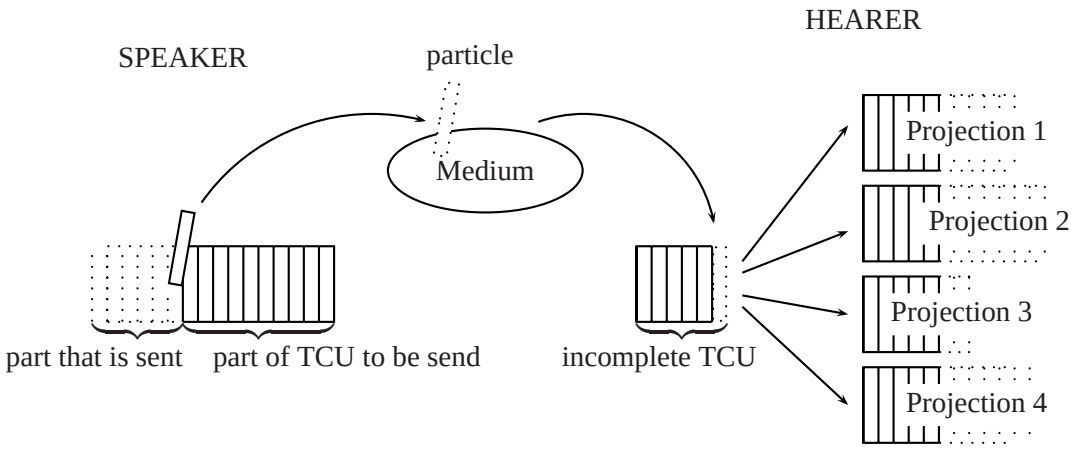

Figure 2: Sending and receiving a TCU

11. *What to listen for*. According to this model, it is natural to listen for particles, and to listen for particles only.

12. *What to do with perceived particles*. In processing a particle, it is first determined who sent it. If no incomplete TCU of the sender is found, then the sender has apparently started to utter a new TCU, and a new TCU is created by the receiver locally. Either way, the particle is pasted to the end of TCU. Then, the updated TCU is projected onto a complete TCU.

    Interpretation amounts to semantically analyzing the TCU, which in our present model comes down to locating attributes such as sender, addressee, contents, intonation, and gesture and contents and extracting them. Further, part of interpreting a TCU amounts to verifying whether (from the perspective of the receiver), the last particle of the TCU has been perceived.

    Projections are produced basically by guessing a possible completion of the TCU's contents, based on existing agent names and the language with the help which the agents communicate. Currently, extrapolation is done simply by returning the first match. For example, a TCU with contents "cab" when extrapolated against language { ..., "buzz", "buzzing", "cabin", "cabdriver", "cabinet", ..., "dab", "dabbler", ... } will yields a complete TCU with contents "cabin".

4 Implementation in Ruby

This section describes an implementation of Sacks *et al.*'s turntaking protocol. This implementation follows the specifications as described above.

For the implementation we selected the programming language Ruby. This language can be considered as a kind of crossbreed between the practical string processing language Perl and the somewhat academic object-oriented language Smalltalk. Ruby is created around 1993 by Yukihiro Matsumoto [20], and is still developed under his responsibility. We choose for Ruby because it is a modern object-oriented scripting language that we consider as extremely well suited for rapid prototyping. Because

of its readability, some even advocated Ruby as executable pseudo code.[3] Although this may be a bit over the top, it clearly points out an important feature of Ruby.

Below are stripped-down versions of some of the routines we implemented. Not all routines are listed because we like to give a global impression, rather than a painstakingly detailed description of the entire implementation. The complete code and documentation is available via [40]. This URL also has an online demo of the implementation.

4.1 Medium

---

routine **Main loop**

```
1: loop
2:    what-just-has-been-said := what-is-said-now # what is said now becomes history
3:    what-is-said-now := empty # to be filled with utterances of speakers
```

The following routine is the publishing link between agents and the medium.

---

routine **Put**(agent, particle)

```
1: what-is-said-now := what-is-said-now ∪ { particle }
```

The following routine is the retrieval link between agents and the medium.

---

routine **Get-each**: Particle

```
1: foreach particle ∈ what-just-has-been-said do
2:    yield particle
```

The yield statement is typical to Ruby. CfÅppendix B on page 34.

4.2 Turn-constructional unit

TCU's are implemented according to the descriptions in the design section of this paper. The most important routine working on TCU's is "Emit". This routine emits subsequent particles of a TCU. For example, if t is TCU with contents "Cat," then subsequent calls of "t.Emit" yield particles with contents "C," "a," "t," and "nil" afterwards.

---

routine **Emit**: Particle

```
1: particle := nil
2: if index ≤ length-of-current-TCU then
3:    if in-neighborhood-of-trp? then
4:       addressee := addressee-of-current-TCU
5:    else
```

[3] A web search with keyword "Ruby" should give enough leads. See also `http://www.ruby-lang.org/`. The de-facto reference is what aficionado's call "the pickaxe book" [35].

```
 6:       addressee := nil
 7:    sender := sender of current TCU
 8:    contents := the index-th element of contents of current TCU
 9:    particle = new Particle with addressee, sender and contents
10: return particle
```

A TCU is updated by pasting the particle's contents to the contents of the TCU. If the particle's addressee is defined, then the addressee of the TCU becomes the particle's addressee. In our implementation, an incomplete TCU is in the neighborhood of TRP if its index is less than two characters from the TCU's end.

The following routine can be considered as the inverse of "Emit".

---

routine **Absorb**(particle)

```
1: extend contents of TCU with contents of particle
2: if addressee of particle is defined then
3:    addressee := addressee of particle
```

4.3 Particle

A particle is implemented as a pure data-structure. (In OO-speak: as a class without methods.) There are no routines for manipulating particles.

4.4 Agent

4.4.1 Main loop

Each agent enters a main loop, such that it is concerned with listening and speaking at every cycle.

---

routine **Main-loop**

```
1: loop
2:    listen # listen first, and listen unconditionally
3:    if speaking then
4:       speak
5:    else
6:       consider-speaking
```

A disadvantage of the current implementation is that each listen-act must be (or actually: is) followed by precisely one speech-act. In more advanced implementations, the listen and speech acts would likely be implemented in separate threads, like

```
thread
    loop { listen }
thread
    loop { if speaking then speak else consider-speaking }
```

or even

thread
    loop { listen }
thread
    loop { evaluate }
thread
    loop { speak }

A downside of implementing separates modules in parallel threads is that it introduces additional complications such as synchronization, coordination and the allocation management of shared resources (fair allocation of time to prevent thread starvation, and mutual exclusion of shared data to prevent corruption of data and deadlock). These are difficult issues that demand additional programming.

We believe that our experiments have shown that a sequential implementation is close enough to Sack's original protocol.

4.4.2 Evaluation Routines

Besides listening, non-speaking participants are busy with other activities. In our current model, the only other activity while being silent is to consider when it is appropriate to start speaking again.

---

routine **Consider-speaking**

```
1: if shall-I-start-speak? then
2:     start-speaking
```

The following routine implements the criterium that determines when a party should starts speaking and might therefore be considered as the heart of Sacks *et al.*'s rule set.

---

routine **Should I start speaking?** : Boolean

```
1: if someone-other-than-me-is-addressed then
2:     return false # the rules say I should remain quiet
3: else # there is a general pause, or I am addressed, or no one is addressed
4:     self-select? shortest-distance-to-trp # I look for an opportunity to break in.
```

The apparent simplicity of the pseudo-code is somewhat deceptive, for two reasons. The first reason is that the actual routine in Ruby is, as all actual agent routines are, interspersed with statements that print out what the agent actually is doing:

```
def shall_I_start_speaking?
   if someone_other_than_me_is_addressed?
      r "Someone is addressed but it's not me"
      false
   else # the rules say I should remain quiet
      if others_are_silent?
         r "Others are silent"
      elsif am_I_addressed?
```

```
            r "I am addressed"
         else
            r "No one is addressed"
         end
         self_select? shortest_distance_to_trp
      end
   end
```

A second reason is that, although, Sacks *et al.*'s original rule set is more complicated than the routine displayed here, it turns out that when this rule set is analyzed, Sacks *et al.*'s rule set can be reduced to a simple if-then-else statement. (Cf. preceding routine "Should I start speaking?")

---

routine **Self-select**

1: $U = \text{TalkFactor}/(1.0 + \text{distance-to-nearest-TRP}^2)$
2: **return** true-with-a-probability-of(u)

If an agent has decided to start speaking, a new TCU is fabricated and the agents starts to speak:

---

routine **Start speaking**

1: current-TCU := fabricate-Turn-constructional-unit
2: speaking := true
3: speak

The routine that is concerned with the fabrication of a turn-constructional unit determines what to say, whether to speak to someone, and whether to mention that person.

---

routine **Fabricate-Turn-constructional-unit**: Turn-constructional unit

1: **if** I-know-name-of-at-least-one-other-speaker **then**
2: **if** true-with-a-probability-of(AddressOtherFactor) **then**
3: to := select-other
4: **if** true-with-a-probability-of(MentionOtherFactor) **then**
5: contents := to
6: **else**
7: contents := select-language-element
8: **else**
9: to := nil
10: contents := select-language-element
11: **else** # I do not know the name of other speakers.
12: to := nil
13: contents := select-language-element
14: from := my-name
15: **return** a new Turn-constructional unit with attributes "from," "to," and "contents".

4.4.3 Speaking routines

The global rule with speaking is to continue, provided not too many other speakers have the floor.

---

routine **Speak**

```
1: if shall-I-continue-speaking? then
2:     emit-particle # continue speaking
3: else
4:     stop-speaking
```

The decision to continue depends on the so-called *urgency to continue*, V. If V is determined, the agent continues speaking with a probability of V.

---

routine **shall-I-continue-speaking?**: Boolean

```
1: X := number-of-agents-speaking # if X = 1 there is only one speaking, which must be me
2: V := 1.0/X^0.5
3: return true-with-a-probability-of(V)
```

If a participant has decided to continue to speak, it emits one particle of its current TCU. Depending on whether a directive is emitted dictating that it should non-verbally address (look at) the addressee, this participant actually non-verbally addresses the addressee.

---

routine **Emit-particle**

```
1: if particle = current-TCU.emit then # the current-TCU was not yet uttered completely
2:     if to = particle.to then
3:         look at participant with ID "to"
4:     else
5:         look at no one in particular
       put particle in the air
6: else # Entire TCU has been uttered.
7:     stop-speaking
```

---

routine **Stop speaking**

```
1: speaking := false
```

4.4.4 Listening routines

The software module that is concerned with listening is the largest of the three modules. This is explained in Section 3.4.3 on page 14.

```
1: foreach particle ∈ Medium do
2:     process particle
```

```
routine Process(particle)
1: sender := particle.sender # who said it?
2: if perceived-TCU-of[sender] does not exist then
3:     perceived-TCU-of[sender] := a new empty TCU
4: update perceived-TCU-of[sender] with particle
5: reconstructed-TCU-from[sender] := project perceived-TCU-of[sender]
6: interpret reconstructed-TCU-from[sender]
```

```
routine Project(Turn-constructional unit): Turn-constructional unit
1: contents := TCU.contents
2: extrapolation := extrapolate-to-language-element(contents) || extrapolate-to-name(contents) ||
   contents
3: return TCU with contents replaced by extrapolation
```

4.5 Results

Ideally, our Ruby implementation should exhibit all manifestations that are listed in Table 1 on page 5. Thus, speaker change should recur; overwhelmingly, one speaker should talk at a time, and so forth.

Appendix C on page 35, shows the output of a Ruby run of Sacks *et al.*'s turntaking protocol (Table 2 on page 7) and additional rules as described at the outset of Section 4. In particular, we see that phenomena nr. 1-7,9, 11, 12, and 14 of Table 1 on page 5 are manifest. Especially phenomenon 14 (the existence of repair mechanisms) is interesting here. The mechanism behind Ruby causing phenomenon 14 is further explained at the outset of Section 4. Phenomena that are notoriously lacking are 8, (varying content of what is said), 10 (varying number of participants) and 13 (linguistic variety of TCU's). Phenomenon 10 can easily be incorporated by creating agents on the fly, so we see no problem here. The absence of Phenomena 10 and 13 is more serious, due to the lack of a grammatical structure and a content of what is said in our simulation. On the other hand, we considered the solution of language-related problems beyond the scope of our assignment. Moreover, the turn-taking behavior does not depend on either the grammatical structure or the content; it only depends on that part of the turn structure that is expressed by TCU's and TCP's.

We conclude that we were able to implement Sacks *et al.*'s turn-taking protocol in Ruby with relatively little effort. The challenge is now to lift, or incorporate this implementation in dedicated agent-oriented programming languages such as 3APL.

5 Implementation in 3APL

For the implementation in 3APL we made use of the 3APL development platform [10]. This platform supports communication by message passing. Each of the participating agents is modeled by one 3APL agent. Since 3APL does not support reading and writing to a shared data space, we programmed a dedicated 'air' agent that synchronizes the communication. For each round, the air agent receives

messages from all agents that are talking, puts them in a list sorted by the agent names, and re-distributes this list among all other agents. Rounds are labeled with an index, to make sure that messages arrive in the same order they were sent.

Alternatively, we could have chosen to let the agents communicate through a file for reading and writing, but since 3APL lacks constructs for mutual exclusion to a resource, we figured this solution would be too cumbersome. The most natural solution, to have all agents send messages to all other agents directly, would be computationally too expensive. Moreover, it would mean a lot of redundant message administration for each single agent. We believe the use of a dedicated 'air' agent presents a good modeling compromise.

5.1 Synchronization

Initially, we did not synchronize messages in a round. However, this made it very hard to maintain in what order messages were send. In the synchronized version of the algorithm, however, a deadlock will occur. If all agents start speaking at exactly the same time, and realize that their speaking interferes with the other agents speaking, they all cease to speak. This conforms to the rules of Sacks et al. So we need a way to break the deadlock. There are several possible solutions.

A first solution, would be to use a time stamp. The agent whose message arrives only a fragment of second earlier is announced the winner, and acquires the exclusive right to speak. However, in our implementation this would mean that agent a would always be allowed to continue, because agent programs are executed in alphanumerical order. Instead we could have a lottery, and assign time stamps and therefore a winner by chance. This lottery can be organized by the 'air' agent.

A second solution was implemented in Ruby. It uses two numerical values: *urgency to self select*, which depends on the *TalkFactor*, a value that is part of an agent's personal character, and *urgency to continue*, a numerical value that depends on the number of other agents speaking. Both of these factors influence the probability that the agent will decide to self-select or to continue. This is a distributed solution. In the 3APL implementation, we selected the first solution.

5.2 3APL Features

As we pointed out in the introduction, 3APL is developed as a agent-based programming language, that allows the use of higher level mental concepts. In particular, an agent program consists of two data-structures that determine the internal state of an agent, the *beliefbase* and the *goalbase*, along with a set of *capabilities* that implement ways of altering the beliefs and of sending and receiving messages. Messages are conceived of as speech acts. For example,

```
Send(I, Addressee, Performative, Content)
```

is the action of sending a message with identifier `I`, to `Addressee`, of which the speech act type is `performative` and the content is `Content`. Clearly, it is a bit odd to use this high-level mechanism for the transmission of low-level particles, such as words or characters. Nevertheless, it is the most convenient way to simulate interaction in 3APL. To structure capabilities into plans, there are so called *practical reasoning rules*. These are production rules of the form `Goal <- Guard | Body`, that operate on the goalbase. The head of the rule, `Goal`, expresses a goal that will be achieved by executing the rule. The `Guard` condition, tests if the rule is compatible with the current beliefs

```
PROGRAM "harry"
CAPABILITIES:   {} Add(Fact) {bel(Fact)}
BELIEFBASE:     me(harry), you(sally)
GOALBASE:       BEGIN
                    hello();
                    talk()
                END
RULEBASE:
hello() <- you(You) AND NOT sent(V, You, greet, AGreeting ) |
                Send(0, You, greet, hello(You) ),
talk() <- you(You) |
                BEGIN
                  Send(0, You, talk, bla(bla) );
                  listen()
                END,
listen() <- received(V, You, talk, Content) |
                BEGIN
                  Add(Content);
                  talk()
                END.
```

Figure 3: Example of 3APL agent program

of the agent. If so, the rule is executed, which means that the head of the rule is substituted in the current goalbase with the body of the rule. Otherwise, another rule is selected. The `Body` contains a sequence of capabilities, or references to other goals.

Figure 3 shows the source code of an example 3APL agent. If two of these agents are executed in parallel, they will start by saying “hello” and continue to send each other “bla(bla)” messages.

5.3 Participants

Part of the protocol depends on knowledge of the participants of who the other participants to the dialogue are. Only these agents qualify as potential speakers. In human face-to-face dialogue this mutual contact is often determined by relative distance, the setting, a room for example, and by mutual nods and glances. In an automated setting, a special facility can be designed to maintain the participant list.

The 3APL agent development platform supports a so called Agent Management System (AMS). This system maintains general information about agents that are present, with the services that they offer. At the beginning of any interaction, agents may register with the AMS, providing information on the services they offer. Agents may also query the AMS for other agents that are present and that offer some particular service.

In the current implementation, the set of agents remains constant during the dialogue. In the future, the AMS registry service can be used to model a dynamic participants list, so that participants may enter or exit the dialogue while it is going on.

5.4 Medium

Instead of having all agents sending messages to each other, we chose to use a central 'air' agent to collect and re-distribute the messages. The air agent is structured as follows. It will collect all messages from the agents in a single round. A round is identified with an integer. If an agent did not sent anything this is simply not added to the round. After collecting all emissions, a round is sorted by the time stamp. In case of interference the utterance of the speaker who started to speak first, will be at the head of the round, and can therefore be easily identified to be the 'winner'. The round is then broadcasted to all other agents.

---

routine **Air Agent**

```
1: pre: knows all agents
2: i := 0; round := [] ;
3: while  not end do
4:    foreach agent do
5:       if a message indexed with i was received then
6:          add (time stamp, agent, message) to round;
7:    sort round by time stamp;
8:    broadcast round to all agents;
9:    i := i+1;round := []
```

The list called 'round' captures exactly the function of the 'what was said' data-structure of Section 3. Because this data-structure is renewed every round, the air is clearly evanescent.

Note moreover that this receiving and re-distribution process is general and re-usable. In our implementation of the turn-taking protocol, messages will in fact be 'chunks' of a turn-constructional unit. But other applications could be the re-distribution of announcements. Therefore, the *agent type*, the abstract structure, exemplified by the air agent can be used to compose other multi-agent systems. We believe that the use of design patterns and agent types facilitates multi-agent system development.

5.5 Turn-constructional unit

3APL does not use structured data types. Instead, it uses Prolog terms, which may be constructed from basic atoms or numbers, and any functor with a number of arguments, each of which is a term in themselves. There is no strict typing discipline in 3APL or Prolog. The programmer is responsible for type-checking arguments of functions. A special kind of nested term produces a list data-structure, which is used instead of arrays or vectors in a conventional language. By definition, a list is either empty, `[]`, or of the form `[H|T]` consisting of an element `H`, called the head, followed by another list `T`, called the tail.

Using such data-structures, the most natural way to represent turn constructional units is by a 3-tuple, of which a list of consecutive particles is the main element. In addition, TRPs are indicated by a list of numbers. Each number refers to the particle after which a break is possible. In the example below the breaks are as follows: "Jan loopt / op straat / met een hoed / en een stok". In this example, breaks coincide with phrase boundaries suggested by Dutch grammar. Compare: *Jan is walking / in the street / with a hat / and a stick.* The last field indicates the person explicitly addressed. If no one is addressed in particular, this field is nil.

```
// tcu( +Particles, +TCPnrs, +Addressed)
tcu( [ jan, loopt, op, straat, met, een, hoed, en, een, stok],
     [ 2,4,7,10 ],
     nil)
```

While the agent is speaking, it maintains a belief about the current state of the TCU it is uttering. The following excerpt shows the code for emitting one particle of a TCU. If the list to be emitted is empty, the agent ceases to speak.

```
speak() <- utt([], TRP, Addr) |       // stop condition TCU=[]
               BEGIN
                  DelUtt();           // remove current utterance
                  Del(speaking);      // change status values
                  DelDesire(to_speak)
               END,
speak() <- utt([H|T], TRP, Addr) |    // recursive call, TCU=[H|T]
               BEGIN
                  emit(Addr,H);       // emit a particle
                  UpdUtt(T, TRP, Addr) // update current utterance
               END,
```

5.6 Agent

The general process structure of the agent is similar to the Ruby implementation. There is a continuous listen-evaluate-speak cycle, during which the agent listens to the messages from other agents, evaluates them and decides whether to speak, continue to speak or remain silent, and finally speaks if so decided.

5.6.1 Main loop

For the individual agents we used a recursive structure to model the main loop. We could have chosen a while-loop as in the case of the air agent.

```
main() <- NOT bel(end) |
    BEGIN
        listen();
        evaluate();
        main()
    END,

main() <- bel(end) | SKIP,
```

5.6.2 Listening routines

An interesting aspect of 3APL is the way the listening module is implemented. Listening is a typical *reactive* process: the agent waits for an event, and if some event occurs, the agents reacts by some appropriate behavior. The 3APL practical reasoning rules are particularly suitable for modeling such reactive behavior. Triggers are represented by a guard condition. If the condition is false, no rule with

that trigger will fire, until the guard condition becomes true. Here the listening rule is triggered by a `received` fact being added to the beliefbase that indicates that a message was received. These routines are very similar to the routines of the 'air' agent.

```
listen() <- received(Ch,air,round,m(Round)) |  // wait for next step
        BEGIN
            DelWinner(); DelAddr(); DelAtTRP() // clear status var's
            Del(received(Ch,air,round,m(Round)));
            length(Round,N)?;                  // Update Nr Speaking
            UpdNrSpeaking(N);
            UpdIndex(Ch);
            IF NOT (Round=[]) THEN
                BEGIN
                    // winner at head of round
                    head(Round,u(T,A,m(Addr,Str)))?;
                    AddWinner(A);
                    AddAddr(Addr)   // winner addresses next speaker
                END;
            listen_all(Round);
        END,
```

Apart from information about the complete round of utterances, such as the winner and the number of speakers, there is also information in the utterances themselves. This must be processed too. In particular, if the project function finds that the speaker is now at a TRP, this is noted.

```
listen_all(L) <- (L = []) | SKIP,  // for all utterances in round

listen_all(L) <- (L = [M|Ms]) |
            IF M = u(Stamp,A,m(Addr,Str)) THEN
                BEGIN
                   IF NOT me(A) THEN ProcessSaid(A,Str);
                   listen_all(Ms)
                END,

ProcessSaid(A,Str) <- said(A,L) |
        BEGIN
            append(L,[Str],L1)?;
            UpdSaid(A,L1);
            // observe TRP by projection
            IF project_trp(L1) THEN AddAtTRP(A)
        END,
```

5.6.3 Evaluation routines

These routines implement the decision of the agent to speak, or stop speaking. Note that a decision to continue to speak is made implicitly, if there is no decision to stop. The decision to stop speaking is made on the basis of the current number of speakers and the 'winner': the speaker who started to speak first. The decision to start speaking, is made on the basis of the number of speakers, whether the agent itself is addressed by the current speaker, and wether the speaker is now at a TRP.

```
decide()<- bel(speaking) |
        BEGIN
            IF nr_speaking(N) AND (N>0) AND me(Me) AND NOT winner(Me)
                THEN stop_speak()
            ELSE speak()
        END,

decide()<- NOT bel(speaking) |
        BEGIN
            IF desire(to_speak) AND
               ( nr_speaking(0) OR
                                              // I am addressed
                 ( me(Me) AND addr(Me) AND at_trp(Sp) ) OR
                 ( addr(nil) AND at_trp(Sp))) // nobody else addressed
               THEN start_speak()
            ELSE silent()
        END,
```

5.6.4 Speaking routines

Most speaking routines are concerned with bookkeeping the current utterance. If the utterance is empty, the agent loses its desire to speak, and deletes some other status variables. Otherwise it emits the utterances particles one by one, recursively.

```
    speak() <- utt([], TRP, Addr) | // ready
                  BEGIN
                     DelUtt(); Del(speaking); DelDesire(to_speak);
                     Add(end)
                  END,

    speak() <- utt([H|T], TRP, Addr) |
                  BEGIN
                     emit(Addr,H);
                     UpdUtt(T, TRP, Addr)
                  END,

    emit(Addr,Str) <- index(I) | Send(I, air, emit, m(Addr,Str)),
```

5.7 Results

Just as for the Ruby implementation, we expect to find that the 3APL implementation produces turn-taking behavior that exhibits the manifestations of Table 1 on page 5. In Appendix A on page 33 we have included an excerpt of one of the log files of a 3APL run.

Observing this and similar other runs, we find that indeed, speaker change occurs (manifestation 1), and that most of the time, one agent is speaking (manifestation 2). Therefore this protocol largely solves the allocation of the exclusive speech channel. Overlap (manifestation 3) does occur. To observe manifestation 4, that transitions with no gap or overlap are common, but that transitions wit ha slight gap or overlap also occur a lot, we have insufficient data. A longer simulation would need to be conducted, with more random elements. Manifestation 5 - 7, about the fact that turn order, turn

size, and length are not fixed, can be observed. Just like for the Ruby implementation the content of what is said is in fact fixed in advance. However, this is not crucial for the turn-taking behavior, as the simulation only depends on the structure of the turn-constructional units, and not on the content. Manifestation 9, that the relative distribution of turns is not specified in advance, can be observed. In the 3APL implementation it follows from the use of a time-stamp when several parties want to start talking simultaneously. Manifestation 10, that the number of parties can vary, is partly observed. The implementation works for any number of agents. At the moment, no agents may enter or exit the dialogue dynamically, but this can in principle be changed using the agent platform. Manifestation 11, that talk can be continuous or discontinuous is observed. Manifestation 12 and 14, about explicit turn allocation techniques (speaker selects next), and repair mechanisms, can both be observed. Again, just like for the Ruby implementation, we allow no linguistic variability in the turn-constructional units.

6 Evaluation

The evaluation will be performed with regards to three criteria: effectiveness, efficiency, and the appropriateness of the test case.

6.1 Effectiveness

Both 3APL and Ruby were capable of modeling the turn-taking protocol in enough detail, to produce satisfactory simulation results. In both cases, some conceptual adaptations were necessary. Ruby is well-equipped for parallel processing, and allows interaction trough a shared data space. 3APL had to resort to the implementation of a special air agent, which collects and re-distributes messages. On the other hand, 3APL agents are true autonomous agents. They are separate programs, that run independently. In the future, the AMS provided by the 3APL platform may be used to handle the entry or exit of participants to a dialogue. Ruby agents are rather like processes, which are dependent on the environment they are deployed in.

6.2 Efficiency

Ruby was much more efficient, both for the programmer implementing the protocol, and for the system, running the implementation. 3APL is a research prototype that combines features of Prolog and Java. The way the Prolog engine is called within the java shell, makes it extremely slow to run, when there are several agents active.

Moreover, the combination of Prolog and Java has lead to some idiosyncracies in the syntax of 3APL, which make it hard to write a program without errors. Naturally, at this stage of the development of the language debugging tools for 3APL are minimal.

The Ruby language on the other hand is fun to use, and allows a lot of flexibility. This makes it suitable as a tool for building fast prototypes.

6.3 Test case

With hindsight, the test case we selected, was indeed a multi-agent case study, but was not very suited for the testing of the higher-level mental attitudes of 3APL agents. Although we have demonstrated

that it is at least possible, it appears that the mental attitudes are not the most natural programming metaphor for capturing low-level real-time interaction aspects. For such aspects, a string based language like Ruby is much more suitable.

7 Conclusions

3APL is indeed a practical programming language, in which it is possible to program a complex multi-agent case study. In particular, the agent metaphor can be used to good effect for modeling autonomous entities, such as participants in a dialogue. An agent platform containing agent management services, can supplement a mechanism for maintaining a dynamic participants list. Reactive behavior can be programmed in a natural way, using practical reasoning rules which a guard condition that serves as an event trigger.

However, the development of 3APL is far from complete. The syntax needs to be re-evaluated. The internal deliberation cycle should be changed so that the guard of a rule is tested just before the execution, and not earlier. This will solve some of the inexplicable time delays between receiving a message and processing it. We believe a true event-based deliberation cycle would be a better choice.

Ruby is a clever high-level programming language, that allows the programmer a lot of flexibility. Its code is easily understandable, and can be used as a kind of pseudo-code. Ruby has mechanisms for parallel execution and mutual exclusion. This makes Ruby suitable for the development of prototypes of distributed systems.

References

[1] Gregory Aist. Expanding a time-sensitive conversational architecture for turn-taking to handle content-driven interruption. In Robert H. Manell and Jordi Robert-Ribes, editors, *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP'98)*, page paper number 928, 1998.

[2] C. Barker and D. Galasinski. *Cultural Studies and Discourse Analysis: A dialogue on language and identity*. London, Sage, 2001.

[3] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: A FIPA-compliant agent framework. In *Proceedings of PAAM'98*, pages 97–108, 1999.

[4] Niels Ole Bernsen, Hans Dybkjaer, and Laila Dybkjaer. *Designing Interactive Speech Systems. From First Ideas to User Testing.* Springer-Verlag, Berlin, 1998.

[5] F.M.T. Brazier, C.M. Jonker, and J. Treur. Compositional design and reuse of a generic agent model. *Applied Artificial Intelligence Journal*, 1999.

[6] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, 27:365–389, 2002.

[7] Herbert H. Clark. *Using Language*. Cambridge University Press, Cambridge, 1996.

[8] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[9] M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation: An approach illustrated using the 3APL language. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, 2003.

[10] M. Dastani, J. van der Ham, and F. Dignum. Communication for goal directed agents. In Marc-Philippe Huget, editor, *Communication in Multiagent Systems - Agent Communication Languages and Conversation Policies*, LNCS 2650, pages 239–252. Springer Verlag, Berlin, 2003.

[11] Virginia Dignum. *A Model for Organizational Interaction, based on Agents, founded in Logic*. PhD thesis, University of Utrecht, 2003.

[12] N. Fay, S. Garrod, and J. Carletta. Group discussion as interactive dialogue or serial monologue: The influence of group size. *Psychological Science*, 11(6):487–492, 2000.

[13] Michael Fisher. A survey of Concurrent MetateM – the language and its applicatuions. In D.M. Gabbay and H.J. Ohlbach, editors, *Temporal Logic: proceedings of the First International Confeence*, volume LNAI 827, pages 480–505. Springer Verlag, Verlin, 1994.

[14] D.E. Hiebeler. The Swarm simulation system and individual-based modeling. In *proceedings of Decision Support 2001: Advanced Technology for Natural Resource Management*, Santa Fe Institute, 94-12-065, 1994.

[15] K.V. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[16] G. Houghton and D. Isard. Why to speak, what to say and how to say it. In P. Morris, editor, *Modelling Cognition*, pages 249–267. Wiley, 1987.

[17] Nick R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[18] M. W. Jorgenson and L. J. Phillips. *Discourse Analysis as Theory and Method*. Thousand Oaks, Sage, 2002.

[19] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[20] Yukihiro Matsumoto. *Ruby in a nutshell*. O'Reilly, 2001.

[21] Scott Moss, Helen Gaylard, Steve wallis, and Bruce Edmonds. SDML: A multi-agent language for organizational modelling. *Computational and mathematical Organization Theory*, 4(1):43–70, 1998.

[22] Emiliano G. Padilha and Jean Carletta. A simulation of small group discussion. In Johan Bos, Mary Ellen Foster, and Colin Matheson, editors, *Proceedings of the 6th Workshop on the Semabntics and Pragmatics of Dialogue (Edilog'02)*, pages 117–124. University of Edinburgh, Edinburgh, 2002.

[23] N. Phillips and C. Hardy. *Discourse Analysis: Investigating processes of social construction*. London, Sage., 2002.

[24] Richard Power. The organisation of purposeful dialogues. *Linguistics*, 17:107–152, 1979.

[25] A. S. Rao. Agentspeak(l): BDI agents speak out in a computable language. In Van de Velde W and J.W. Perram, editors, *Agents breaking Away: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume LNAI 1038, pages 42–55. Springer Verlag, Berlin, 1996.

[26] Anand S. Rao. Integrated agent architecture: Execution and recognition of mental-states. In *Intelligent Agent Systems: Theoretical and Practical Issues*, volume 1087 of *Lecture notes in computer science*, pages 159–173. Springer-Verlag, Berlin, 1996.

[27] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the International Workshop on Knowledge Representation (KR91)*, pages 473–484. Morgan Kaufmann, San Mateo CA, 1991.

[28] H. Sacks, E.A. Schegloff, and G. Jefferson. A simplest systematics for the organisation of turn-taking for conversation. *Language*, 50:696–735, 1974.

[29] Y. Shoham. Agent-oriented programming. *Aretificial Intelligence*, 60(1):51–92, 1993.

[30] Ronnie W. Smith. An evaluatiuon of strategies for selectively verifying utterance meanings in spoken natural language dialog. *Human Computer Studies*, 48:627–647, 1998.

[31] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-agent Systems*, 7(1-2):29–48, 2003.

[32] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, New York, 2002.

[33] Gerard Tel. Distributed control algorithms for ai. In G. Weiss, editor, *MultiAgent Systems: A Modern Approach to Distibuted Artificial Intelligence*, chapter 13, pages 539–580. Cambridge, MA: MIT Press, 1999.

[34] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge U.K., 2000.

[35] David Thomas and Andrew Hunt. *Programming Ruby: The Pragmatic Programmer's Guide*. Addison-Wesley, 2000. Aka "the Pickaxe book" (because the cover shows a pickaxe mining rubies). Available online at http://www.rubycentral.com/book/.

[36] Rebecca Thomas. The PLACA agent programming language. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents*, volume LNAI 890, pages 355–370. Springer Verlag, Berlin, 1995.

[37] David Traum and Peter Heeman. Utterance units in spoken dialogue. In E. Maier, M. Mast, and S. LuperFoy, editors, *Dialogue Processing in Spoken Language Systems*, volume LNCS 1236, pages 125–140. Springer-Verlag, Heidelberg, 1997.

[38] M.B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in dribble: from beliefs to goals with plans. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, 2003.

[39] Marko Verbeek. 3APL as programming language for cognitive robots. Master's thesis, ICS, Utrecht University, 2002.

[40] Gerard A.W. Vreeswijk. Code, documentation, and online demo's on turntaking. `http://www.cs.uu.nl/~gv/code/turntaking`, 2003.

[41] M. Wetherell, S. Taylor, and S. Yates, editors. *Discourse Theory as Practice*. London, Sage, 2001.

[42] Michael Wooldridge. *An Introduction To MultiAgent Systems*. John Wiley and Sons, Chchester, 2002.

[43] Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

A Output of a 3APL run

The following table shows an excerpt of one of the log files of a 3APL run. In round 0, all agents start speaking together. As agent a is the first, (time stamp 1600), it continues. At the end of round 2, agent b notices that agent a is at a TRP, and since he was addressed by a, is entitled to interrupt. b tries again in round 5.

ID	*Sender*	*Addressee*	*Performative*	*Content*
0	agent_c	air	emit	m(nil,marie)
0	agent_b	air	emit	m(nil,piet)
0	agent_a	air	emit	m(agent_b,jan)
1	air	agent_a	round	m([u(1600,agent_a, m(agent_b,jan)), u(1862,agent_b, m(nil,piet)), u(2193,agent_c,m(nil,marie))])
1	air	agent_b	round	m([u(1600,agent_a, m(agent_b,jan)), u(1862,agent_b, m(nil,piet)), u(2193,agent_c,m(nil,marie))])
1	air	agent_c	round	m([u(1600,agent_a, m(agent_b,jan)), u(1862,agent_b, m(nil,piet)), u(2193,agent_c,m(nil,marie))])
1	agent_a	air	emit	m(agent_b,loopt)
2	air	agent_a	round	m([u(4682,agent_a,m(agent_b,loopt))])
2	air	agent_b	round	m([u(4682,agent_a,m(agent_b,loopt))])
2	air	agent_c	round	m([u(4682,agent_a,m(agent_b,loopt))])
2	agent_a	air	emit	m(agent_b,op)
2	agent_b	air	emit	m(nil,piet)
3	air	agent_a	round	m([u(6498,agent_a,m(agent_b,op))])
3	air	agent_b	round	m([u(6498,agent_a,m(agent_b,op))])
3	air	agent_c	round	m([u(6498,agent_a,m(agent_b,op))])
4	air	agent_a	round	m([u(7318,agent_b,m(nil,piet))])
4	air	agent_b	round	m([u(7318,agent_b,m(nil,piet))])
3	agent_a	air	emit	m(agent_b,straat)
4	air	agent_c	round	m([u(7318,agent_b,m(nil,piet))])
5	air	agent_a	round	m([u(9408,agent_a,m(agent_b,straat))])
5	air	agent_b	round	m([u(9408,agent_a,m(agent_b,straat))])
5	air	agent_c	round	m([u(9408,agent_a,m(agent_b,straat))])
5	agent_a	air	emit	m(agent_b,met)
6	air	agent_a	round	m([u(6348,agent_a,m(agent_b,met))])
6	air	agent_b	round	m([u(6348,agent_a,m(agent_b,met))])
6	air	agent_c	round	m([u(6348,agent_a,m(agent_b,met))])
5	agent_b	air	emit	m(nil,piet)
7	air	agent_a	round	m([u(7832,agent_b,m(nil,piet))])
7	air	agent_b	round	m([u(7832,agent_b,m(nil,piet))])
6	agent_a	air	emit	m(agent_b,een)
7	air	agent_c	round	m([u(7832,agent_b,m(nil,piet))])
8	air	agent_a	round	m([u(9756,agent_a,m(agent_b,een))])
8	air	agent_b	round	m([u(9756,agent_a,m(agent_b,een))])
7	agent_b	air	emit	m(nil,rent)
8	air	agent_c	round	m([u(9756,agent_a,m(agent_b,een))])

B The yield statement in Ruby

The Ruby implementation of Sacks *et al.*'s protocol uses a special kind of return-statement, called `yield`, which requires explanation.

In Ruby, the yield statement calls a block, optionally passing it one or more parameters. For example,

```
def callBlock
  yield
  yield
end

callBlock { puts "In the block" }
```

produces:

```
In the block
```

The code in the block (which is `puts "In the block"`) is executed twice, once for each call to yield.

Some people like to think of the association of a block with a method as a kind of parameter passing, but it is better to think of the block and the method as coroutines, which transfer control back and forth between themselves.

It is possible to provide parameters to the call to yield. These will be passed to the block. Within the block, the names of the arguments to receive these parameters are listed between vertical bars ("`|`"). For example,

```
class Array
   def iterate
      for element in self do # "self" is the array itself
         yield element # replace this line by the body of
      end # the calling block, passing "element"
   end
end

[ 'cat', 'dog', 'horse' ].iterate do |animal|
   puts animal # body of calling block
end
```

Output:

```
cat
dog
horse
```

The `yield` statement in Ruby differs significantly from the `yield` statement in other languages such as Python. For more details, see the documentation of the appropriate languages.

C Output of a Ruby run

The following table shows the output of a random run with five participants, using language

$$\mathcal{L} = \{\text{“hello”}, \text{“hi”}, \text{“hey”}, \text{“howdy”}, \text{“holy-moly”}, \text{“hail”}, \text{“hark”}\}.$$

A list of words hardly counts as a language of course, but it fulfill our needs for the purpose of turntaking. Notice that all words have a common prefix, which ensures that agents have some work to do in projecting initial utterances of others. In this connection, $\mathcal{L}$ is organized such that the words most frequently uttered come first.

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I say something. What shall I say? No one has spoken yet, so names I do not know. Let me say ”holy-moly” to no one in particular. Looking at no one in particular, I utter ”h”.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I stay quiet.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I stay quiet.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I say something. What shall I say? No one has spoken yet, so names I do not know. Let me say ”hark” to no one in particular. Looking at no one in particular, I utter ”h”.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I say something. What shall I say? No one has spoken yet, so names I do not know. Let me say ”hi” to no one in particular. Looking at no one in particular, I utter ”h”.
h			**h**	**h**
From Dexter I project ”hello”. From Elisha I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I stop speaking.	From Antoine I project ”hello”. From Dexter I project ”hello”. From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Antoine I project ”hello”. From Dexter I project ”hello”. From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Antoine I project ”hello”. From Elisha I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at no one in particular, I utter ”a”.	From Antoine I project ”hello”. From Dexter I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at no one in particular, I utter ”i”.
			a	**i**

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Dexter I project ”hail”. From Elisha I project ”hi”. He completes one TCU. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hark” to Dexter. Looking at no one in particular, I utter ”h”.	From Dexter I project ”hail”. From Elisha I project ”hi”. He completes one TCU. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Dexter I project ”hail”. From Elisha I project ”hi”. He completes one TCU. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Elisha I project ”hi”. He completes one TCU. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I stop speaking.	From Dexter I project ”hail”. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I stop speaking.
h				
Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at no one in particular, I utter ”a”.	From Antoine I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.	From Antoine I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.	From Antoine I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.	From Antoine I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.
a				
Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Dexter, I utter ”r”.	From Antoine I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I say something. What shall I say? Let me say ”hail” to Dexter. Looking at no one in particular, I utter ”h”.	From Antoine I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I stay quiet.	From Antoine I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I stay quiet.	From Antoine I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I stay quiet.
r [to Dexter]	**h**			

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Benet I project ”hello”. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I continue. Looking at Dexter, I utter ”k”.	From Antoine I project ”hark”. He addresses Dexter. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I continue. Looking at no one in particular, I utter ”a”.	From Antoine I project ”hark”. He addresses Dexter. From Benet I project ”hello”. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Antoine I project ”hark”. He is addressing me. From Benet I project ”hello”. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 1. My urgency to speak is $0.50/(1+1^2) = 0.25$. I decide I stay quiet.	From Antoine I project ”hark”. He addresses Dexter. From Benet I project ”hello”. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.
k [to Dexter]	**a**			
From Benet I project ”hail”. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I stop speaking.	From Antoine I project ”hark”. He addresses Dexter. He completes one TCU. Examining whether I should continue speaking. Including myself, there are 2 speaking. My urgency to continue is 1/sqrt(2) = 0.71. I decide I continue. Looking at Dexter, I utter ”i”.	From Antoine I project ”hark”. He addresses Dexter. He completes one TCU. From Benet I project ”hail”. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Antoine I project ”hark”. He is addressing me. He completes one TCU. From Benet I project ”hail”. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Antoine I project ”hark”. He addresses Dexter. He completes one TCU. From Benet I project ”hail”. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.
	i [to Dexter]			
From Benet I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Dexter, I utter ”l”.	From Benet I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hail”. He is addressing me. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 1. My urgency to speak is $0.50/(1+1^2) = 0.25$. I decide I stay quiet.	From Benet I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.
	l [to Dexter]			

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Benet I project ”hail”. He addresses Dexter. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. ¡B¿I’m done uttering TCU with contents ”hail”¡/B¿. Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hi” to Elisha. Looking at Elisha, I utter ”h”.	From Benet I project ”hail”. He addresses Dexter. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hail”. He is addressing me. He completes one TCU. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Benet I project ”hail”. He addresses Dexter. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.
	h [to Elisha]			
From Benet I project ”hello”. He addresses Elisha. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Elisha, I utter ”i”.	From Benet I project ”hello”. He addresses Elisha. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hello”. He addresses Elisha. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hello”. He is addressing me. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.
	i [to Elisha]			
From Benet I project ”hi”. He addresses Elisha. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. ¡B¿I’m done uttering TCU with contents ”hi”¡/B¿. Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Benet I project ”hi”. He addresses Elisha. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hi”. He addresses Elisha. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Benet I project ”hi”. He is addressing me. He completes one TCU. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hi” to Benet. Looking at Benet, I utter ”h”.
				h [to Benet]

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Elisha I project ”hello”. He addresses Benet. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hello”. He is addressing me. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Elisha I project ”hello”. He addresses Benet. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hello”. He addresses Benet. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Benet, I utter ”i”.
				i [to Benet]
From Elisha I project ”hi”. He addresses Benet. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hi”. He is addressing me. He completes one TCU. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I stay quiet.	From Elisha I project ”hi”. He addresses Benet. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hi”. He addresses Benet. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. ¡B¿I’m done uttering TCU with contents ”hi”¡/B¿. Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is 0.50/(1+0^2) = 0.50. I decide I say something. What shall I say? Let me say ”hey” to Antoine. Looking at no one in particular, I utter ”h”.
				h
From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is 0.50/(1+4^2) = 0.03. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Antoine, I utter ”e”.
				e [to Antoine]

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Elisha I project ”hello”. He is addressing me. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 3. My urgency to speak is $0.50/(1+3^2) = 0.05$. I decide I stay quiet.	From Elisha I project ”hello”. He addresses Antoine. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hello”. He addresses Antoine. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hello”. He addresses Antoine. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Antoine, I utter ”y”.
				y [to Antoine]
From Elisha I project ”hey”. He is addressing me. He completes one TCU. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	From Elisha I project ”hey”. He addresses Antoine. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hey”. He addresses Antoine. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Elisha I project ”hey”. He addresses Antoine. He completes one TCU. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. ¡B¿I’m done uttering TCU with contents ”hey”¡/B¿. Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.
Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hail” to Dexter. Looking at no one in particular, I utter ”h”.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hello” to Antoine. Looking at no one in particular, I utter ”h”.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I stay quiet.	Examining whether I should start speaking. Others are silent. Deciding whether to self-select. Distance from nearest TRP is 0. My urgency to speak is $0.50/(1+0^2) = 0.50$. I decide I say something. What shall I say? Let me say ”hark” to Antoine. Looking at no one in particular, I utter ”h”.
h		**h**		**h**

Antoine	*Benet*	*Caspar*	*Dexter*	*Elisha*
From Caspar I project ”hello”. From Elisha I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at no one in particular, I utter ”a”.	From Antoine I project ”hello”. From Caspar I project ”hello”. From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.	From Antoine I project ”hello”. From Elisha I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at no one in particular, I utter ”e”.	From Antoine I project ”hello”. From Caspar I project ”hello”. From Elisha I project ”hello”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 4. My urgency to speak is $0.50/(1+4^2) = 0.03$. I decide I stay quiet.	From Antoine I project ”hello”. From Caspar I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at no one in particular, I utter ”a”.
a		**e**		**a**
From Caspar I project ”hello”. From Elisha I project ”hail”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I continue. Looking at Dexter, I utter ”i”.	From Antoine I project ”hail”. From Caspar I project ”hello”. From Elisha I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I stay quiet.	From Antoine I project ”hail”. From Elisha I project ”hail”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I stop speaking.	From Antoine I project ”hail”. From Caspar I project ”hello”. From Elisha I project ”hail”. Examining whether I should start speaking. No one is addressed. Deciding whether to self-select. Distance from nearest TRP is 2. My urgency to speak is $0.50/(1+2^2) = 0.10$. I decide I stay quiet.	From Antoine I project ”hail”. From Caspar I project ”hello”. Examining whether I should continue speaking. Including myself, there are 3 speaking. My urgency to continue is 1/sqrt(3) = 0.58. I decide I stop speaking.
i [to Dexter]				
Examining whether I should continue speaking. There is one speaking, and that’s me. My urgency to continue is 1/sqrt(1) = 1.00. I decide I continue. Looking at Dexter, I utter ”l”.	From Antoine I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Antoine I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.	From Antoine I project ”hail”. He is addressing me. Examining whether I should start speaking. I am addressed. Deciding whether to self-select. Distance from nearest TRP is 1. My urgency to speak is $0.50/(1+1^2) = 0.25$. I decide I stay quiet.	From Antoine I project ”hail”. He addresses Dexter. Examining whether I should start speaking. Someone is addressed but it’s not me. I decide I stay quiet.
l [to Dexter]				